ЭКЗАМЕНАЦИОННЫЕ ВОПРОСЫ

по дисциплинам "**Объектно-ориентированные технологии программирования и стандарты проектирования**" и "**Объектно-ориентированное программирование**"

для студентов специальностей 1-40 01 01 «Программное обеспечение информационных технологий» и 1-40 05 01 «Информационные системы и технологии»

Реализация абстракции в ООП. Абстрактные классы и интерфейсы в Java

1) Абстракция (абстрактный уровень или слой данных и приложения).
2) Абстрактное поведение (абстрактный метод).
3) Основная концепция абстрактного класса в Java.
4) Содержимое абстрактного класса.
5) Отличие абстрактного класса от обычного класса в Java.
6) «Множественное наследование» в Java.
7) Концепция и основная идея отделения интерфейса от реализации.
8) Интерфейс – ещё один пользовательский тип данных в Java.
9) Объявление (декларирование), конструкторов, блоков инициализации, полей и методов внутри интерфейса, их модификаторы.
10) Объявление методов с модификатором *final*, *abstract* или *static* внутри интерфейсов.
11) Расширение интерфейсов через наследование.
12) Правила реализация (имплементации) интерфейсов классами.
13) Отличия интерфейсов от абстрактных классов в Java и их использования.
14) Создание объектов интерфейсов и абстрактных типов данных.

Перечисляемый тип в Java (since JDK 5.0)

15) Основная концепция перечисляемого типа данных в Java.
16) Общие характеристики и возможности перечисления в Java.
17) Класс *java.lang.Enum*, основное состояние и поведение данного класса.
18) Сходство и отличия перечисляемого типа данных от класса в Java.
19) Интерфейсы и перечисление.
20) Конструкторы перечисления. Переопределение методов перечисления.
21) Статическое импортирование и перечисление.
22) Перечисление и оператор множественного выбора *switch*.
23) Оператор *instanceof* в Java.
24) Принципиальное различие ссылочных переменных, которые имеют тип абстрактного класса, обычного класса, интерфейса, перечисления и массива.

SOLID и GRASP принципы

25) Постоянная величина (константа) любого программного обеспечения (ПО).
26) Признаки плохого кода (проекта): жёсткость (*rigidity*), хрупкость (*fragility*), неоправданная сложность (*needless complexity*), неопределённость или смутное представление (*opacity*), *needless repetition* и т.д.
27) Чистый код. Рефакторинг как часть жизненного цикла ПО.

28) Основополагающие принципы, базирующие на фундаментальных концепциях ООП и лежащие в основе SOLID и GRASP принципов (пример, «*инкапсулируйте то, что изменяется*», «*предпочитайте композицию наследованию*», «*программируйте на уровне интерфейса*», «*стремитесь к слабой связанности взаимодействующих объектов*» и т.д.).
29) SOLID принципы – пять основных (фундаментальных) принципов проектирования в ООП.
30) Концепция и примеры принципа единственной ответственности (*Single Responsibility Principle, SRP*) при грамотной разработки единиц программного кода (в частности, функций, методом, классов, модулей и пакетов).
31) Концепция и примеры принципа открытости/закрытости (*Open-Closed Principle, OSP*).
32) Концепция и примеры принципа подстановки Барбары Лисков (*Liskov Substitution Principle, LSP*).
33) Концепция и примеры принципа разделения интерфейса (*Interface Segregation Principle, ISP*).
34) Концепция и примеры принципа инверсии зависимостей (*Dependency Inversion Principle, DIP*).
35) GRASP (*General Responsibility Assignment Software Patterns)* – девять общих шаблонов распределения ответственностей, используемые в ООП для решения общих задач по назначению ответственностей классам и объектам:
    a) информационный эксперт (*Information Expert*);
    b) создатель (*Creator*);
    c) контроллер (*Controller*);
    d) слабая степень связности (*Low Coupling*);
    e) высокое зацепление (*High Cohesion*);
    f) полиморфизм (*Polymorphism*);
    g) чистая выдумка (*Pure Fabrication*);
    h) посредник (*Indirection*);
    i) устойчивость к изменениям (*Protected Variations*).

Приёмы объектно-ориентированного проектирования. Шаблоны проектирования

36) Шаблоны проектирования и общая их классификация (структурные, порождающие и поведенческие шаблоны проектирования). Преимущества и недостатки использования шаблонов проектирования.
37) Поведенческий шаблон проектирования «Стратегия» (*The Strategy Pattern*): основная цель, концепция и компоненты шаблона.
38) Поведенческий шаблон проектирования «Итератор» (*The Iterator Pattern*): основная цель, концепция и компоненты шаблона.
39) Структурный шаблон проектирования «Декоратор» (*The Decorator Pattern*): основная цель, концепция и компоненты шаблона.
40) Структурный (архитектурный) шаблон проектирования MVC: основная цель, концепция и компоненты шаблона (модель (*model*), представление (*View*), контроллер (*Controller*)).

41) Порождающий шаблон проектирования Абстрактная фабрика (*The Abstract Factory Pattern*): основная цель, концепция и компоненты шаблона. Простая фабрика (*The Simple Factory Pattern*).
42) Порождающий шаблон проектирования Строитель (*The Builder Pattern*): основная цель, концепция и компоненты шаблона.
43) Порождающий шаблон проектирования Одиночка (*The Singleton Pattern*): основная цель, концепция и компоненты шаблона.

Параметризация (обобщение) типов в Java – Java Generic (since JDK 5.0)

44) Основная концепция и основные предпосылки появления параметризированных типов в Java.
45) Общая форма объявления параметризированного класса, интерфейса, конструктора и метода, diamond-синтаксис, новшества в JDK 7.0.
46) Параметризированные типы и класс Object.
47) Параметризированные типы и типы данных.
48) Параметризированный класс с двумя и больше параметрами типа.
49) Языковая безопасность типов с использованием параметризированных типов, а также явное приведение типов.
50) Правило именования типов параметров в параметризированных типах и методах.
51) Базовые типы и унаследованный код. Иерархии параметризированных классов и интерфейсов.
52) Переопределение методов в параметризированном классе.
53) Приведение параметризированных типов.
54) Процесс «стирания» при работе с параметризированными типами как механизм автоматической обработки исходного кода.
55) Ограничения, присущие параметризированным типам: получение экземпляра параметризированного типа, ограничения на статические компоненты, ограничения на параметризированные массивы и исключения.

Библиотека контейнеров Java Collections Framework (since JDK 5.0)

56) Дайте определения Java-контейнерам (коллекциям) и перечислите основные концепции их использования и преимущества по сравнению с другими типами данных.
57) Какие данные могут хранить контейнеры?
58) Опишите общую архитектуру (картину) и иерархию библиотеки Java Collection Framework (JCF): раздел интерфейсов, раздел абстрактных и конкретных классов-реализаций, раздел классов-алгоритмов.
59) В какой библиотеки размещается JCF?
60) С помощью какой абстракции в JCF можно перебрать элементы любого контейнера или как осуществить доступ к элементам коллекции?
61) Где и как используется паттерн Iterator, какова его реализация в JCF?
62) Зачем нужны интерфейсы Iterable и Iterator, как они работают?
63) В каких случаях может быть выброшено ConcurrentModificationException?
64) Какой интерфейс необходимо реализовать в пользовательском классе и как, чтобы объект данного класса можно было использовать с модифицированным в JDK5.0 циклом for (в простонародий, foreach)?

65) Опишите основное поведение любой коллекции (перечислите все методы, которые декларирует интерфейс Collection).

66) Опишите стандартные интерфейсы-поведения JCF: List, Set (SortedSet, NavigableSet), Queue, Deque, Map (SortedMap, NavigableSet) и др.

67) Для чего нужны абстрактные классы JFC: AbstractCollection, AbstractSequentialList, AbstractList, AbstractMap, AbstractSet, AbstractQueue и др.

68) Опишите основные классы-реализации поведения списков (имплементация интерфейса List), их внутреннюю архитектуру, а также их преимущества и недостатки: ArrayList, LinkedList и др.

69) Когда лучше использовать ArrayList и LinkedList, а в каких случаях разумно использовать массив?

70) Перечислите способы перебора элементов в списках. Зачем для списков нужен дополнительный интерфейс ListIterator?

71) Чем отличается интерфейс Iterator от интерфейса ListIterator?

72) Опишите основные классы-реализации поведения множества (расширение и имплементация интерфейса Set), их внутреннюю архитектуру, а также их преимущества и недостатки: HashSet, LinkedHashSet, TreeSet и др.

73) Как и при помощи какой логики поддерживается уникальность объектов соответствующими реализациями интерфейса Set?

74) В какой из реализаций поведения Set не допускается в качестве значения добавлять null-ссылку?

75) Опишите основные классы-реализации алгоритмов абстрактных структур данных типа очереди, стека, дека и т.д. (расширение и имплементация интерфейса Queue), их внутреннюю архитектуру, а также их преимущества и недостатки: PriorityQueue, ArrayDeque и др.

76) Опишите основные классы-реализации поведения контейнеров, поддерживающих отображение, т.е. контейнеры, которые хранят элемент в виде пары «ключ-значение» (расширение и имплементация интерфейса Map), их внутреннюю архитектуру, а также их преимущества и недостатки: HashMap, LinkedHashMap, TreeMap и др.

77) Что будет, если в Map положить два значения с одинаковым ключом?

78) Охарактеризуйте потокобезопасные контейнеры, которые существовали с JDK 1.0 ещё до выхода JCF и основное предназначение каждого: Vector, Stack, BitSet, Dictionary, Hashtable, Properties и др.;

79) Чем отличается ArrayList от Vector и Stack?

80) Чем Hashtable отличается от HashMap? На сегодняшний день Hashtable deprecated, как все-таки использовать нужную функциональность?

81) Опишите утилитные классы для работы с контейнерами (массивами): Collections, Arrays и др.

82) Как задается порядок следования объектов в контейнерах? Как отсортировать коллекцию?

83) Какой необходимо реализовать интерфейс для того, чтобы объекты пользовательского класса могли быть, к примеру, отсортированы в своём естественном виде (natural ordering)?

84) В чем разница между интерфейсами Comparable и Comparator?

85) Как получить не модифицируемую (только для чтения) коллекцию?
86) Опишите архитектуру и поведение шаблона Стратегия (The Strategy Pattern). Когда и где его применяют?

Основы потоков ввода-вывода в Java

87) Что такое файл и атрибуты файла?
88) Что такое файловая система? Основные функции файловой системы?
89) Какие типы файлов в общем случае существуют в компьютерном мире?
90) Опишите основную концепцию использования потоков ввода-вывода (*streams*) в Java для организации процессов хранения, записи, чтения и обмена данными между программами.
91) Что физически представляет собой поток?
92) В каких стандартных Java-пакетах сосредоточены основные реализации потоков ввода-вывода?
93) На базе какого структурного шаблона проектирования базируется система классов и объектов ввода-вывода в Java?
94) Какие разновидности потоков существуют в Java?
95) Какую организацию имеют все средства ввода-вывода в Java на самом низком уровне?
96) Опишите общее поведение (интерфейсы), которым обладают почти все потоки ввода-вывода в Java.
97) Опишите основные классы исключительных ситуаций для поддержки работы системы ввода-вывода. Какое исключение является центральным для данной системы?
98) Какова основная концепция байтовых потоков ввода-вывода?
99) Опишите базовые классы, задающие основную абстракцию поведения байтовых потоков.
100) Какова основная концепция символьных потоков ввода-вывода?
101) На базе какой кодировки реализована работа символьных потоков?
102) Опишите базовые классы, задающие основную абстракцию поведения символьные потоков.
103) В чём отличия байтовых и символьных потоков ввода-вывода, а в чём их схожесть?
104) Опишите укрупнённо UML-диаграмму иерархии основных классов системы ввода-вывода в Java и их взаимосвязи.
105) Как шаблоны проектирования упрощают и(или) улучшаю профессиональное общение между разработчиками?
106) Назначение и работа с классом *Scanner* (Java API).
107) Что такое сериализация?
108) Что такое десериализация?
109) Стандартные средства для реализации сериализации и десериализации в Java.

Основы параллельного (многопоточного) программирования в Java. Потоки выполнения

110) Концепция многопоточности, процессы и потоки, ключевые понятия.

111) Модель параллельного (многопоточного) программирования в Java, понятие потока выполнения с точки зрения внутреннего устройства JVM.
112) Приведите общее описание сущности-потока в Java и его характеристики (свойства).
113) Главный (дочерний) поток. Получение ссылки на поток для его управления.
114) Жизненный цикл сущности-потока, состояния потока.
115) Основные подходы для программной реализации потоков в Java: класс Thread и интерфейс Runnable. Преимущества и недостатки.
116) Характеристики потоков: идентификатор, название, приоритет, группа, состояние и т.д.
117) Планирование и управление потоками.
118) Группы потоков.
119) Приоритет потока.
120) Поток-демоны. Отличия интерактивных потоков от потоков-демонов (фоновых потоков).

Основы синхронизации потоков в Java. Использование библиотеки java.util.concurrency

121) Условия состояния гонки потоков (*racing condition*).
122) Концепция синхронизации и блокировки потоков в Java.
123) Полная картина жизненного цикла потока с учётом состояния блокировки при синхронизации.
124) Понятие монитора и других объектов, применяемых для синхронизации доступа.
125) Реализация синхронизации на уровне языка Java.
126) Работа с ключевым словом *synchronized*, синхронизированные методы, синхронизированные блоки.
127) Взаимная блокировка (*deadlock*), условия возникновения и пути её решения.
128) Атомарность выполнения операции. Использование ключевого слова *volatile* в Java.
129) Понятия блокирующей и неблокирующей синхронизации.
130) Основные интерфейсы и реализации для организации неблокирующей синхронизации библиотеки *java.util.concurrent.locks*: *Lock, ReentrantLock* и т.д.